LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-CONF-464132

Extending ALE3D, an Arbitrarily Connected hexahedral 3D Code, to Very Large Problem Size (U)

A. L. Nichols

December 16, 2010

NECDC 2010
Los Alamos, NM, United States
October 18, 2010 through October 22, 2010

Disclaimer

Extending ALE3D, an Arbitrarily Connected hexahedral 3D Code, to Very Large Problem Size (U)

Albert L. Nichols, III

Lawrence Livermore National Laboratory, Livermore, California, 94551

This poster will describe the reasons, methods, issues associated with extending the ALE3D code to run problems larger than 700 million elements. (U)

Introduction

As the number of compute units increases on the ASC computers, the prospect of running previously unimaginably large problems is becoming a reality. In an arbitrarily connected 3D finite element code, like ALE3D, one must provide a unique identification number for every node, element, face, and edge. This is required for a number of reasons, including defining the global connectivity array required for domain decomposition, identifying appropriate communication patterns after domain decomposition, and determining the appropriate load locations for implicit solvers, for example. In most codes, the unique identification number is defined as a 32-bit integer. Thus the maximum value available is 2^{31}, or roughly 2.1 billion. For a 3D geometry consisting of arbitrarily connected hexahedral elements, there are approximately 3 faces for every element, and 3 edges for every node. Since the nodes and faces need id numbers, using 32-bit integers puts a hard limit on the number of elements in a problem at roughly 700 million.

The first solution to this problem would be to replace 32-bit signed integers with 32-bit unsigned integers. This would increase the maximum size of a problem by a factor of 2. This provides some head room, but almost certainly not one that will last long. Another solution would be to replace all 32-bit int declarations with 64-bit long long declarations. (long is either a 32-bit or a 64-bit integer, depending on the OS). The problem with this approach is that there are only a few arrays that actually need to extended size, and thus this would increase the size of the problem unnecessarily. In a future computing environment where CPUs are abundant but memory relatively scarce, this is probably the wrong approach. Based on these considerations, we have chosen to replace only the global identifiers with the appropriate 64-bit integer.

The problem with this approach is finding all the places where data that is specified as a 32-bit integer needs to be replaced with the 64-bit integer. that need to be replaced. In the rest of this paper we describe the techniques used to facilitate this transformation, issues raised, and issues still to be addressed.

Scope

Once one has decided that some subset of the data needs to be upgraded from 32-bit to 64-bit integer, one should ask how far such a transformation should extend. From the introduction, at least for ALE3D, it is clear that anything describing a global identifier for node, element, face, and edge needs to be upgraded. For current platforms, one can show that 64-bit integers are not needed to handle the local ids in a single domain. If we have 2^{31} nodes/elements, the memory required to just hold the locations and connectivity is 3 64-bit reals and 8 integers. For a problem just below the 2-billion element level, this equates to a memory requirement of approximately 112 GB using 32-bit integers and 176 GB

with 64-bit integers. This memory would be required on every node of the computer used to solve this class of problem. Current trends in super computer design suggest that this amount of memory per computational unit is going unlikely for the foreseeable future. Thus, we did not considered migrating the local index data to 64-bit integers.

ALE3D also supports shell, beam, and point loads. In general, these are add-ons to the standard 3D mesh that are associated with surface issues. Thus, for a several billion element problem, we would expect on the order of a couple of million shell elements, and even fewer beam and point loads. Thus, we felt it was not necessary to convert these element types at this time.

Methodology

The first issue associated with upgrading the global identifier is how to represent it in the code. Also, changing the data type of the global identifier has an impact on several areas of the code. These include memory management, standard I/O, and interactions with third party libraries.

Data type

We had several requirements for this data transformation. First, we required was that the change of the global identifier be a compile time choice, as not all packages/third party libraries are capable of handling the new data size. We defined a compile time definition **GLOBALID_IS_64BIT** that is set to 0 for 32-bit integers and 1 for 64-bit. When set to 0, we were able to preserve all of our test process for 32-bit integers. We could then also add appropriate failure statements if the user requested a feature not supported in the 64-bit version. With some modification of the testing process, this allowed us to use the 64-bit version on our standard 32-bit test suite.

Another requirement was that we wanted mismatches in the data type to be caught early by the compiler. This would reduce the amount of testing required to insure correct implementation. Our initial concept was to define a typedef that would be an int by default but would be a long long int for the large problem size builds. This works to some extent, except that by default the C and C++ compilers will do automatic promotion between various types of integers. To get around this issue, we defined a C++ globalID that has a single data member of the appropriate size. All type conversion construction operators are declared explicit to prevent incidental conversion between 32-bit and 64-bit ints.

As part of the process of separating out globalIDs from standard integers, several routines that had been implemented to handle integer data had to be duplicated to handle globalID data instead. There were several locations where we would convert a global ID into a local ID using the same data storage locations. For these situations, we had to either keep both local and globalID storage, or would NULL out the pointer to the data type that was currently not being used. The second approach is especially useful when making sure that the correct data type was being accessed.

One issue that made finding errors difficult is the existence of routines like qsort that take a void * pointer. These remove any type checking from the compiler. To find these issues, we created type specific wrappers to qsort that would use the correct compare routine for the data type.

Library Issues

ALE3D depends on several third party libraries. Most of these libraries do not require globalID information, and so were unaffected by the transformation.

SILO: Silo is used to write our restart and plot files. Silo already supported the long int data type. Unfortunately, the long data type is 32-bits on a 32-bit system, and 64-bits on a 64-bit system. To insure

portability of our restart files, we asked the Silo team to add support for long long int.

MPI: The MPI specification already includes **MPI_LONG_LONG_INT** for 64-bit integers. We defined **ALE_MPI_GID** to be the appropriate MPI data type, depending on the build type.

The matrix solution libraries FEI, from Sandia, HYPRE, and FEMSTER, both from LLNL, require globalIDs to properly assign the matrix equation numbers. Since they do not yet support 64-bit equation numbers, we had to disable their use. However, since we do have an internal solver, we did propagate the changes there. Specific down grade operators were constructed to insure that the original API to these routines was maintained without having to exclude them from the compile.

Several of the external mesh decomposers are known not to scale well with large processor counts. The packages have not been converted to handle 64-bit integers.

External File Formats

Most of the external files that ALE3D interacts with do not have long long support. Thus, data needs to be down converted when written out to these formats or up converted when reading from them. The file formats include OVERLINK, which is used to exchange data between problem runs, SAMI, which is used to describe the mesh, and TrueGrid, which is also used to describe the mesh. The restart and plot files have been upgraded to handle long long data, and VisIt should be able to display long long data for global node ids.

Development Environments

One thing that can be used to make the process of converting data types is a good integrated development environment (IDE). A good IDE will provide the ability to quickly jump to compiler errors, allowing the developer to see and fix the issues quickly. The IDE also provides a mechanism to jump to the prototype of a function. This allows the developer to duplicate an integer based routine with one set up for globalIDs. Finally, a good IDE makes it easy to jump to all instances of a function, allowing them to be updated as need before it is caught by the compiler.

Testing

ALE3D has over 600 tests that are required to be run before one is allowed to update the code. In order to pass these tests, one must achieve a bit-wise binary comparison between the new restart file and the baseline, with the exception of certain designated field. Since some of the fields are globalIDs, when run in 64 bit mode, those fields had to be added to the list of designated differences.

Finally, since some of the packages are not enabled in 64 bit globalID mode, we added a standard message to the code output to indicate that that package was not available and thus the problem could not run. These conditions were not considered a failure for the test, and so the test suite could therefore complete with any failures.

Acknowledgements

The author would like to thank the members of the ALE3D group, especially Jim Reus and Bill Arrighi for helpful comments and advice. I would also like to thank Mark Miller for the changes made to the Silo package to support long long integers. This work was performed under the auspices of the United States Department of Energy by the Lawrence Livermore National Laboratory under Contract No. DE-AC52-07NA27344